Focused Crawling through Reinforcement Learning

Miyoung Han[1,3], Pierre-Henri Wuillemin[4], and Pierre Senellart[1,2,3]

[1] LTCI, Télécom ParisTech
[2] DI ENS, ENS, CNRS, PSL University
[3] Inria
[4] Sorbonne University, CNRS, LIP6, Paris

Abstract. Focused crawling aims at collecting as many Web pages relevant to a target topic as possible while avoiding irrelevant pages, reflecting limited resources available to a Web crawler. We improve on the efficiency of focused crawling by proposing an approach based on reinforcement learning. Our algorithm evaluates hyperlinks most profitable to follow over the long run, and selects the most promising link based on this estimation. To properly model the crawling environment as a Markov decision process, we propose new representations of states and actions considering both content information and the link structure. The size of the state-action space is reduced by a generalization process. Based on this generalization, we use a linear-function approximation to update value functions. We investigate the trade-off between synchronous and asynchronous methods. In experiments, we compare the performance of a crawling task with and without learning; crawlers based on reinforcement learning show better performance for various target topics.

1 Introduction

Focused crawlers are autonomous agents designed to collect Web pages relevant to a predefined topic, for example to build a search engine index. Given a start page (the *seed*), a crawler browses Web pages by exploiting hyperlinks of visited Web pages to find relevant pages. Usually, crawlers maintain a priority queue of new URLs, called the *frontier*. Each new URL is assigned a priority value and URLs are fetched from the queue in decreasing order of priority. Since the focused crawler aims to collect as many relevant pages as possible while avoiding irrelevant pages, the key success factor of crawling systems is how good the scoring policy is.

The priority score is initially based on contextual similarity to the target topic [4, 10], on link analysis measures such as PageRank and HITS [14, 20, 1], or on a combination of both [2, 5]. However, links that look less relevant to the target topic but that can potentially lead to a relevant page in the long run may still be valuable to select. *Reinforcement learning* (RL) enables the agent to estimate which hyperlink is the most profitable over the long run. A few previous studies have applied reinforcement learning to crawling [22, 13, 16, 17], but they require an off-line training phase and their state definitions do not consider the link structure; for example, states are represented with a vector which consists of the existence or frequency of specific keywords.

Hence, we propose a reinforcement learning based crawling method that learns link scores in an online manner, with new representations of states and actions considering

both content information and the link structure. Our method assumes that the whole Web graph structure is not known in advance. To properly model the crawling environment as a Markov decision process (MDP), instead of considering each individual page as a state and each individual hyperlink as an action, we generalize pages and links based on some features that represent Web pages and the next link selection, thus reducing the size of the state–action space. To allow efficient computation of a link score, i.e. *action-value*, we approximate it by a linear combination of the feature vector and a parameter vector. Through this modeling, we can estimate an action value for each new link, to add it to the frontier. As action values computed at different time steps are used in the frontier, we investigate a synchronous method that recalculates scores for all links in the frontier, along with an asynchronous one that only compute those of all outlinks of current page. As an improved asynchronous method, we propose moderated update to reach a balance between action-values updated at different time steps. In experiments, we compare our proposed crawling algorithms based on reinforcement learning and an algorithm without learning.

The paper is organized as follows. Sec. 2 presents some important background. Sec. 3 introduces our algorithm, focused crawling with reinforcement learning. Sec. 4 describes the details of our experiments and shows the performance evaluation of our algorithm. Sec. 5 presents prior work in the literature. Sec. 6 concludes with some further work.

2 Background

Focused Crawler, Topical Locality, and Tunneling. A focused crawler selects from the frontier the links that are likely to be most relevant to a specific topic(s). It aims to retrieve as many relevant pages as possible and avoid irrelevant pages. This process consequently brings considerable savings in network and computational resources.

Focused crawlers are based on topical locality [8, 15]. Pages are likely to link to topically related pages. Web page authors usually create hyperlinks in order to help users navigate, or to provide further information about the content of the current page. If hyperlinks are used for the latter purpose, the linked pages may be on the same topic as the current page and hyperlinks can be useful information for topic-driven crawling. Davison [8] shows empirical evidence of topical locality on the Web. He demonstrates that linked pages are likely to have high textual similarity. Menczer [15] extends the study and confirms the existence of link–content conjecture that a page is similar to the pages that link to it and that of link–cluster conjecture that two pages are considerably more likely to be related if they are within a few links from each other. The author shows that the relevance probability is maintained within a distance of three links from a relevant page, but then decays rapidly.

To selectively retrieve pages relevant to a particular topic, focused crawlers have to predict whether an extracted URL points to a relevant page before actually fetching the page. Anchor text and surrounding text of the links are exploited to evaluate links. Davison [8] shows that titles, descriptions, and anchor text represent the target page and that anchor text is most similar to the page to which it points. Anchor text may be useful in discriminating unvisited child pages.

Although a focused crawler depends on the topical locality, pages on the same topic may not be linked directly and it can be necessary to traverse some off-topic pages to reach a relevant page, called tunneling [3]. When going through off-topic pages, it is needed to decide if the crawl direction is good or not. Bergmark et al. [3] propose a tunneling technique that evaluates the current crawl direction and decides when to stop a tunneling activity. They show tunneling improves the effectiveness of focused crawling and the crawler should be allowed to follow a series of bad pages in order to get to a good one. Ester et al. [11] also propose a tunneling strategy that reacts to changing precision. If precision decreases dramatically, the focus of the crawl is broaden. Conversely, if precision increases, the focus goes back to the original user interest.

Reinforcement Learning. Reinforcement learning [23] is learning from interaction with an environment to achieve a goal. It is a powerful framework to solve sequential decision-making problems. The agent discovers which actions produce the greatest reward by experiencing actions and learns how good it is to take a certain action in a given state over the long-term, quantified by the value of action. Reinforcement learning aims to maximize the total reward in the long run.

The notion of Markov Decision Process (MDP) underlies much of the work on reinforcement learning. An MDP is defined as a 4-tuple $M = \langle S, A, R, T \rangle$ where S is a set of states, A a set of actions, $R : S \times A \to \mathbb{R}$ a reward function, and $T : S \times A \times S \to [0,1]$ a transition function. The reward function returns a single number, a reward, for an action selected in a given state. The transition function specifies the probability of transition from state s to state s' on taking action a. A policy $\pi : S \to A$ maps states to actions that maximize the total reward in the long run. The goal of the agent is to find an optimal policy π^*.

When a learning agent takes an action a in state s, it receives a reward r, and moves to the next state s' choosing an action a' with policy π. In the SARSA algorithms [23], one of the widely used methods, the estimated value of taking action a in state s, denoted $Q(s,a)$, is updated as:

$$Q(s,a) \coloneqq Q(s,a) + \alpha \left[r + \gamma Q(s',a') - Q(s,a) \right]. \tag{1}$$

Here, α is a positive fraction such that $0 < \alpha \leq 1$, the step-size parameter that influences the rate of learning; the discount rate γ ($0 \leq \gamma < 1$) determines the present value of future rewards. The difference between the estimates at two successive time steps, $r + \gamma Q(s',a')$ and $Q(s,a)$, is called the temporal difference (TD) error or the Bellman error.

Linear Function Approximation based on Gradient-Descent Method. In small finite and discrete state spaces, value functions above are represented using a tabular form that stores the state-action values in a table.

However, in many realistic problems, state spaces are infinitely large and it is difficult to represent and store value functions in a tabular form. Thus the tabular methods have to be extended to function approximation methods.

In many cases, the value function of action a in state s is approximated using a linear function with weight vector $\mathbf{w} \in \mathbb{R}^d$ and it is denoted $\hat{q}(s,a,\mathbf{w}) \approx q_\pi(s,a)$. Action a in state s is represented with a real-valued vector of features, called a feature vector,

$\mathbf{x}(s,a) \coloneqq (x_1(s,a), x_2(s,a), \ldots, x_d(s,a))$. Each $x_i(s,a)$ is a function $x_i : S \times A \to \mathbb{R}$ and its value is called a feature of s and a. In linear methods, the weight vector $\mathbf{w}$ has the same number of elements as the feature vector $\mathbf{x}(s,a)$. Then the action-value function is approximated by the inner product between $\mathbf{w}$ and $\mathbf{x}(s,a)$:

$$\hat{q}(s,a,\mathbf{w}) \coloneqq \mathbf{w}^{\mathsf{T}}\mathbf{x}(s,a) = \sum_{i=1}^{d} w_i x_i(s,a). \tag{2}$$

At each time step, we update the weight vector $\mathbf{w}$. The gradient descent methods are commonly used in function approximation. By the gradient descent method, the weight vector $\mathbf{w}$ is changed by a small amount in the direction that minimizes the TD error. The gradient-descent update based on SARSA method is:

$$\mathbf{w}_{t+1} \coloneqq \mathbf{w}_t + \alpha \left[r_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}, \mathbf{w}_t) - \hat{q}(s_t, a_t, \mathbf{w}_t) \right] \nabla \hat{q}(s_t, a_t, \mathbf{w}_t). \tag{3}$$

Priority-Based Value Iteration. Value Iteration (VI) is a dynamic programming algorithm that performs sweeps of updates across the state space until convergence to optimal policies. VI has some inefficiencies because updates are performed on the entire state space at each iteration even though some updates do not change value functions and those updates are not performed in an optimal order. Performing updates on a state after updating its successors can be more efficient than that in an arbitrary order.

Prioritized Sweeping (PS) is a well-known prioritization method introduced by Moore et al. [19]. The principal idea of PS is to update backwards from states that have changed in value to the states that lead into them, i.e. its predecessors. Prioritizing updates has an effect of propagating values backwards from states with a relatively high state-values and it enables to reduce the number of updates. PS maintains a priority queue to order updates and Bellman error is commonly used as the priority metric. PS can start from any state, not just a goal state. If the priority of the state–action pair is greater than a certain threshold, then the pair is stored in the queue with its priority. For each iteration, a state–action pair (s,a) with the highest priority is removed from the queue and its value function is updated. For each predecessor of (s,a), its priority value is computed. If the priority is greater than some threshold then the predecessor is inserted in the priority queue.

Prioritized methods considerably improve the performance of value iteration by reducing the number of updates, but maintaining the priority queue may result in excessive overhead. It is tackled in the literature by prioritizing updates without a priority queue [7], stationary update orders [6, 9], or state space partitioning [24, 25].

3 Focused Crawling and Reinforcement Learning

The goal of focused crawling is to collect as many pages relevant to the target topic as possible while avoiding irrelevant pages because the crawler is assumed to have limited resources such as network traffic or crawling time. Thus, in a sequence of crawling, link selection should not be a random choice.

To achieve the crawling goal, given a page, the agent selects the most promising link likely to lead to a relevant page. Even though a linked page looks less relevant to

the target topic, if it can potentially lead to a relevant page in the long run, it might be valuable to select it. At each time step, the agent has to estimate which hyperlink can lead to a relevant page. It will be a key success factor in crawling if the agent has an ability to estimate which hyperlink is the most profitable over the long run.

Reinforcement learning finds an optimal action in a given state that yields the highest total reward in the long run by the repeatedly interaction with the environment. With reinforcement learning, the optimal estimated value of hyperlinks (actions) are learned as pages (states) are visited. The agent can evaluate if a link selection can yield a long-term optimal reward and selects the most promising link based on the estimation. In this section, we discuss how to model a focused crawling with Reinforcement Learning. Like most focused crawlers, we assume that pages with similar topics are close to each other. Our crawling strategy is based on the topical locality and tunneling technique. We also assume that the whole Web graph structure is not known to the crawling agent in advance.

3.1 Markov Decision Processes (MDPs) in Crawling

To model the crawling environment in an MDP $M = \langle S, A, R, T \rangle$, we define Web pages as states S and direct hyperlinks of a page as actions A. When the crawling agent follows a hyperlink from the current page, a transition from the current page to the linked page occurs and a relevance to the target topic is computed for the linked page to evaluate if the selected hyperlink leads to a page relevant to the target topic or not. The transition function T is the probability of transition from the current page to the linked page on taking the hyperlink. The reward $r \in R$ is a relevance value of the linked page to the given topic. For the next crawling step, the agent selects a hyperlink with the highest estimation value from the newly visited page, and so on.

Before applying the model above to solve our crawling problem, we must consider two issues: first, scalability of state-action space in a reinforcement learning, second, applicability to a crawling task without loss of its inherent process property. For the scalability problem, we reduce a state-action space by generalization presented in this section and update value functions with linear function approximation discussed in Sec. 3.3. For the applicability issue, in order to preserve the original crawling process we prioritize updates, see Sec. 3.2 and 3.3.

In this section, we discuss modeling a crawling task as an MDP. As we mentioned above, we define Web pages as states and direct hyperlinks of a page as actions. However, Web pages are all different, there are a huge amount of pages on the Web, and they are linked together like the threads of a spider's Web. If each single Web page is defined as a state and each direct hyperlink as an action, it makes learning a policy intractable due to the immense number of state-action pairs. Furthermore, in reinforcement learning, optimal action-values are derived after visiting each state-action pair infinitely often. It is not necessary for a crawler to visit the same page several times. Thus, our MDPs can not be modeled directly from a Web graph. Instead, we generalize pages and links based on some features that represent Web pages and the next link selection. By this generalization, the number of state-action pairs is reduced and Web graph is properly modeled in an MDP. Some pages with the same feature values are in the same state.

Some hyperlinks with the same feature values also can be treated as the same action. The features extracted from pages and hyperlinks are presented in the following.

States. A Web page is abstracted with some features of Web pages in order to define a state. The features of a state consists of two types of information. The first one is proper information about the page itself. The second is relation information with respect to surrounding pages. Page relevances to the target topic and to some categories are the current pages's own information. Relevance change, average relevance of parent pages, distance from the last relevant page represent the relation with the pages surrounding the current page. In order to properly obtain the relation information, each unvisited link should retain parent links. The crawling agent is assumed not to know the whole Web graph in advance, thus each link initially does not know how many parents they have but parent information is progressively updated as pages are crawled. When a page is visited, the URL of the current page is added to all outlinks of the page as their parent. Each link has at least one parent link. If a link has many parents, it means that the link is referenced by several pages.

Most features are continuous variables, which we specified with two different indexes discretized into 5 and 6 buckets according to value ranges: 1) the range $[0.0, 0.2]$ by 0, $[0.2, 0.4]$ by 1, ..., $[0.8, 1.0]$ by 4. 2) the range $[0.0, 0.1]$ by 0, $[0.1, 0.3]$ by 1, ..., $[0.9, 1.0]$ by 5. The relevance value is also discretized according to value ranges as above but occasionally the value has to be converted to a Boolean to specify if a page is relevant to a given topic or not. For example, two features, the Average Relevance of Relevant Parents and Distance from the Last Relevant Page, require true/false value regarding to the relevance. To avoid an arbitrary threshold for relevance, we simplify the definition of relevant page as follows: if a crawled page has a tf-idf based relevance greater than 0.0 or simply contains the target topic word, the page is defined to be relevant to the topic.

- **Relevance of Target Topic:** The target topic relevance based on textual content is computed by cosine similarity between a word vector of target topic and that of the current page and it is discretized according to value ranges.
- **Relevance Change of Target Topic:** The current page's relevance to the target topic is compared to the weighted average relevance of all its ancestors on the crawled graph structure. The weighted average relevance of all its ancestors are computed in an incremental manner by applying an exponential smoothing method on the parents pages.

  Before we explain how to calculate the weighted average relevance of its all ancestors, we simply note that in the exponential smoothing method, the weighted average y at time i with an observation x_i is calculated by: $y_i = \beta \cdot x_i + (1-\beta) \cdot y_{i-1}$, where $\beta (0 < \beta < 1)$ is a smoothing factor. The exponential smoothing assigns exponentially decreasing weights on past observations. In other words, recent observations are given relatively more weight than the older observations.

  In our crawling example, if the relevance of a page x is denoted $rl(x)$, then the weighted average relevance of x, $w_{rl}(x)$ is obtained by $w_{rl}(x) = \beta \cdot rl(x) + (1-\beta) \cdot \max_{x' \to x} w_{rl}(x')$.

  If the current page has many parents, i.e. many path from its ancestors, the maximum average among them, $\max_{x' \to x} w_{rl}(x')$, is used for the update. $w_{rl}(x)$ is the weighted average relevance over x and all its ancestors on the crawled graph structure.

Then, we can calculate the relevance change between current page p and $w_{rl}(x)$ where x is a parent of p: $change \leftarrow rl(p) - \max_{x \to p} w_{rl}(x)$.
The change helps to detect how much the relevance of the current page is increased or decreased than the average relevance of its ancestors.
The relevance change to the current page from its ancestors is discretized according to value ranges. With predefined parameters δ_1 and δ_2, the difference within δ_1 is indexed by 0, the increase by δ_2 is indexed by 1, increase more than δ_2 is indexed by 2, decrease by δ_2 is indexed by 3, and decrease more than δ_2 is indexed by 4.

- **Relevances of Categories:** Given a target topic, its related categories in a category hierarchy such as the Open Directory Project (ODP, https://www.dmoz.org/) are properly selected by the designer of the system. For each category, its relevance is calculated by cosine similarity between a word vector of the category and that of the current page. It is discretized according to value ranges.
- **Average Relevance of All Parents:** The average of all parents' relevance is calculated and discretized according to value ranges.
- **Average Relevance of Relevant Parents:** The average of relevant parents' relevance is calculated and discretized according to value ranges.
- **Distance from the Last Relevant Page:** The distance on the crawl path from the last relevant ancestor page to the current page is simply calculated by adding 1 to the parent's distance. If there are many parents, the minimum distance among them is used. The distance value is capped at 9 to keep it within a finite range.

Actions. In order to define actions, all hyperlinks in a Web page are also abstracted with some features in a similar way as pages are. Relevances to the target topic and to some categories are used to predict the relevance of the page that a hyperlink points to. Different from pages, hyperlinks do not have sufficient information to calculate the values. Thus, the URL text, the anchor text and surrounding text of a hyperlink are used to compute. Here, the relevance is not a true relevance but a prediction because it is not possible to know which page will be pointed by a hyperlink before following the link. In order to support the relevance prediction, the average relevances of parent pages are also used as features that represent the relation with the pages surrounding the link. Each hyperlink has at least one parent. If the link is referenced by several pages, it can have many parents. As mentioned above, parent information is progressively updated as pages are crawled and each unvisited link retains parent links. Then, the parent information is used to compute average relevance of parent pages. The features for action are a subset of those of states, namely: Relevance of Target Topic, Relevances of Categories, Average Relevance of All Parents, Average Relevance of Relevant Parents.

The size of a discretized state space is $(10)^3 \cdot (10)^{\text{num of categories}} \cdot 5 \cdot 10$ and the size of action space is $(10)^3 \cdot (10)^{\text{num of categories}}$. For example, if there is just one category, the size of the state space is $5 \cdot 10^5$ and the size of the action space is 10^4.

3.2 MDPs with Prioritizing Updates

In a focused crawl, the agent visits a Web page and extracts all hyperlinks from the page. The hyperlinks are added to the priority queue, called frontier. A link with the highest priority is selected from the frontier for the next visit. The frontier plays a crucial

role in the crawling process. The agent can take the broad view of the crawled graph's boundary, not focusing on a specific area of the whole crawled graph. Unvisited URLs are maintained in the frontier with priority score and therefore, for each iteration, the most promising link can be selected from the boundary of the crawled graph. Thus, the Web crawler can consistently select the best link regardless of its current position.

We use a temporal difference (TD) method of reinforcement learning in order to make crawling agents learn good policies in an online, incremental manner as crawling agents do. In most TD methods, each iteration of value updates are based on an episode, a sequence of state transitions from a start state to the terminal state. For example, at time t, in state s the agent takes an action a according to its policy, which results in a transition to state s'. At time $t+1$ in the successor state of s, state s', the agent takes its best action a' followed by a transition to state s'' and so on until the terminal state. While crawling, if the agent keeps going forward by following the successive state transition, it can fall into crawling traps or local optima. That is the reason why a frontier is used importantly in crawling. It is necessary to learn value functions in the same way as crawling tasks.

To keep the principle idea of crawling tasks, we model our crawling agent's learning with prioritizing the order of updates that is one of value iteration methods to propagate the values in an efficient way. With a prioritized update method, the crawling agent does not follow anymore the successive order of state transitions. Each state-action pair is added to the frontier with its estimated action value. For each time, it selects the most promising state-action pair among all pairs as the traditional crawling agent does.

3.3 Linear Function Approximation with Prioritizing Updates

We have modeled our crawling problem as an MDP and defined features of the states and the actions in Sec. 3.1. Then, we have presented prioritized updates in reinforcement learning to follow the original crawling process in Sec. 3.2. In this section, we discuss how to represent and update action-value functions based on the state and action features defined in Sec. 3.1.

As discussed in Sec. 3.2, the crawling frontier is a priority queue. Each URL in the frontier is associated with a priority value. The links are then fetched from the queue in order of assigned priorities. In our crawling model, we estimate an action value for each unvisited link and add it to the frontier with its action value.

In reinforcement learning, if a state space is small and discrete, the action value functions are represented and stored in a tabular form. But, this method is not suitable for our crawling problem with a large state-action space. Thus, we use a function approximation method, in particular the linear function approximation, to represent action values. The action value function is approximated by linearly combining the feature vector $\mathbf{x}(s,a)$ and the parameter vector $\mathbf{w}$ with Eq. (2). State and action features defined in Sec. 3.1 are used as the components of a feature vector $\mathbf{x}(s,a)$. At each time step, the weight vector $\mathbf{w}$ is updated using a gradient descent method, as in (3). The approximated action-value obtained from Eq. (2) is used as the priority measure.

When we calculate action-values only for the outlinks of the current page with newly updated weights and add them to the frontier, an issue can arise in the scope of state-action pairs regarding computation of action value. This problem is caused from the prioritized order of selecting a link from the frontier. If the agent keeps going forward

by following the successive state transition, it is correct that calculating action values is applied only to the direct outlinks because the next selection is decided from one of the all outlinks. However, in the prioritized order selecting from the frontier, when the weight vector $\mathbf{w}$ is changed, action values of all links in the frontier also have to be recalculated with the new $\mathbf{w}$. We call this *synchronous* method. Recalculating for all links is the correct method but it involves an excessive computational overhead. Otherwise, we can calculate action-value only for outlinks of the current page and/or recalculate all links(actions) in the frontier that are from the current state. The action values of all other links in the frontier are left unchanged. We call this *asynchronous* method. This method does not incur computational overhead but action values of all links in the frontier are calculated at different time steps and make it difficult to choose the best action from the frontier. In experiments, we compare the performance of the two methods.

Since the asynchronous method has an advantage that does not need to recalculate action values of all unvisited links in the frontier, we try to improve the asynchronous method. The problem of asynchronous method is that action values computed in different time steps exist together in the frontier and it can cause a noise in selection. Thus, we reduce the action value differences in the frontier by manipulating weight updates. The TD error is the difference between the estimates at two successive time steps, $r+\gamma\hat{q}(s',a',\mathbf{w})$ and $\hat{q}(s,a,\mathbf{w})$. Updating the error to weights signifies that the current estimate $\hat{q}(s,a,\mathbf{w})$ is adjusted toward the update target $r+\gamma\hat{q}(s',a',\mathbf{w})$. In order to moderate the TD error, we adjust the estimate $\hat{q}(s',a',\mathbf{w})$ by the amount of the TD error when updating weights as follows:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[r + \gamma(\hat{q}(s',a',\mathbf{w}) - \delta) - \hat{q}(s,a,\mathbf{w}) \right] \nabla \hat{q}(s,a,\mathbf{w}) \tag{4}$$

where $\delta = r + \gamma\hat{q}(s',a',\mathbf{w}) - \hat{q}(s,a,\mathbf{w})$. We call this *moderated* update. In fact, this moderated update can be shown to have same effect as reducing the step-size α of the original update by $1-\gamma$. The idea behind the moderated update is to decrease an overestimated action value or to increase an underestimated action value of the update target in order to make a balance between action-values updated at different time steps.

In experiments, we compare the performance of synchronous, asynchronous methods and asynchronous method with moderated update.

Our reinforcement learning for crawling is outlined in Algorithm 1. The crawling task is started with seed pages (lines 5–13). The frontier is filled with (s,a) pairs of all outlinks from the seed pages. A link is extracted from the frontier according to the ε-greedy policy (lines 16–20). With small probability ε, the agent selects a link uniformly at random. Otherwise, it selects greedily a link from the frontier. The agent fetches the page corresponding to the selected link and defines feature values of the newly visited state as described in Sec. 3.1 (line 24). All outlinks in the fetched page are retrieved (line 25). For each outlink, action feature values are defined as described in Sec. 3.1 (line 30). The weight vector $\mathbf{w}$ of linear function approximation is updated based on a reward and feature vectors of the new state returned from the fetch in line 24 (lines 32–39). With the updated weight vector, an estimated action value of each outlink is computed and added to the frontier with the estimated value. If we use the synchronous method, action values of all hyperlinks in the frontier $(l,\cdot,\cdot)$ are recalculated (lines 40–43). With the asynchronous method, hyperlinks $(l',s',\cdot)$ that are from the state s' are

Algorithm 1 Focused Crawling based on Reinforcement Learning

Input: seed links *Seeds*, maximum number of pages to visit *LIMIT_PAGES*
Initialize value-function weights $w \in \mathbb{R}^d$
$B \leftarrow \emptyset$ # contains (s,a) pairs

while *Seeds* is not empty **do**
    Select a link l from *Seeds*
    $s \leftarrow$ Fetch and parse page l
    $L' \leftarrow$ Extract all outlinks of l
    for each $l' \in L'$ **do**
        $(l',s',a') \leftarrow$ Get action features a' of l'
        Add (l',s',a') to (s',a') pair of B with initial Q-value
    end for
end while

while visited_pages $<$ *LIMIT_PAGES* **do**
    if With probability ε **then**
        Select a (s,a) pair uniformly at random from B and select a link (l,s,a) from the pair
    else
        Select a (s,a) pair from B with highest Q-value and select a link (l,s,a) from the pair
    end if
    if l is visited **then**
        continue
    end if
    $r,s' \leftarrow$ Fetch and parse page (l,s,a)
    $L' \leftarrow$ Extract all outlinks of l
    for each $l' \in L'$ **do**
        if l' is visited **then**
            continue
        end if
        $(l',s',a') \leftarrow$ Get action features a' of l'
    end for
    if visited page is relevant **then**
        $w \leftarrow w + \alpha\,[r - \hat{q}(s,a,w)]\,\nabla\hat{q}(s,a,w)$
    else
        Choose a' as a function of $\hat{q}(s',\cdot,w)$ with ε-greedy policy
        $\delta \leftarrow r + \gamma\hat{q}(s',a',w) - \hat{q}(s,a,w)$
        $w \leftarrow w + \alpha\,[r + \gamma\hat{q}(s',a',w) - \hat{q}(s,a,w)]\,\nabla\hat{q}(s,a,w)$ #*original* update
        $w \leftarrow w + \alpha\,[r + \gamma(\hat{q}(s',a',w) - \delta) - \hat{q}(s,a,w)]\,\nabla\hat{q}(s,a,w)$ #*moderated* update
    end if
    for each $(\cdot,\cdot)$ pair $\in B$ **do** #*synchronous* method
        Calculate Q-value of $(\cdot,\cdot)$
        Update $(\cdot,\cdot)$ to B with Q-value
    end for
    for each $(s',\cdot)$ pair $\in L'$ **do** #*asynchronous* method
        Calculate Q-value of $(s',\cdot)$
        Add $(l',s',\cdot)$ to $(s',\cdot)$ pair of B with Q-value
    end for
    visited_pages $\leftarrow$ visited_pages + 1
end while

updated with new action values (lines 44–47). This process repeats until the visit counter reaches the predefined visit limit.

4 Experimental Results

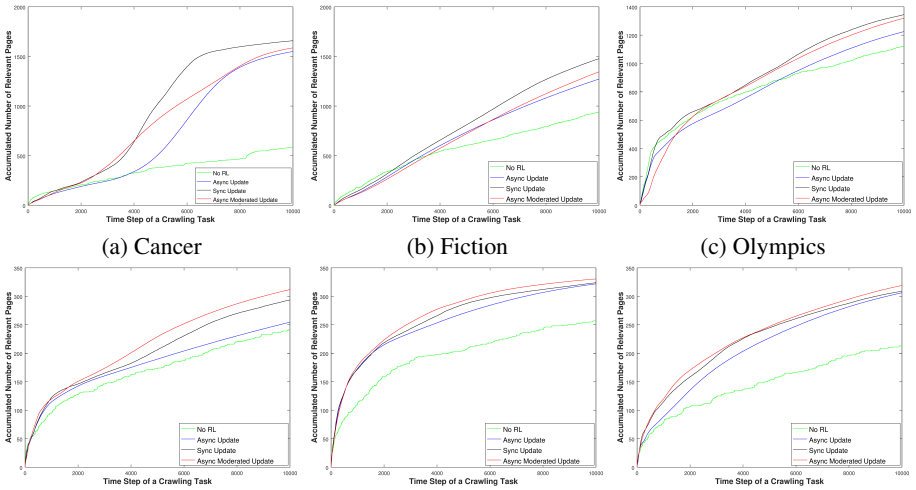
A crawling task starts with a seed page and terminates when the visit counter reaches the predefined visit limit. In our experiments, the limit of the page visit is set to 10,000. For each time step, the agent crawls a page and obtains a reward based on two values: cosine similarity based on tf-idf, and cosine similarity with word2vec vectors pre-trained from https://nlp.stanford.edu/projects/glove/ (w2v). If a crawled page has a tf-idf based relevance greater than 0.0 or simply contains the target topic word, the page is relevant to the target topic and then the agent receives a reward of 30. If a page has a tf-idf based relevance lower than 0.0 but it has a w2v based relevance greater than 0.5 or 0.4, the agent receives a reward of 30 or 20 respectively because the content of such page is rather related to the target topic and could eventually lead to a relevant page. Otherwise, the agent receives a reward -1 per time step.

As a crawling environment, we use a database dump of Simple English Wikipedia provided by the site https://dumps.wikimedia.org/. As target topics to use in our experiments, we choose three topics, Fiction, Olympics, and Cancer, of which relevant pages are fairly abundant and another three topics, Cameras, Geology, and Poetry, of which relevant pages are sparse in our experimental environment. For each target topic, a main page corresponding to the topic is used as a seed page. In all experiments, parameter settings for learning are $\varepsilon = 0.1$, discount rate $\gamma = 0.9$, and step size $\alpha = 0.001$. For topic Olympics and Fiction, step size α is set to 0.0005.

Each feature of state and action is specified with two indexes discretized 5 and 6 buckets. In the case of 'Relevance Change of Target Topic' feature, it is discretized into 5 buckets. The parameter δ_1 and δ_2 used for discretizing its value are set to 0.1 and 0.3 respectively and smoothing factor β is set to 0.4. The number of features are different depending on a target topic because categories vary according to the target topic. In our experiments, categories are empirically pre-selected based on the Open Directory Project (ODP, http://www.dmoz.org, http://curlie.org/), an open directory of Web links. The ODP is a widely-used Web taxonomy that is maintained by a community of volunteers. Among the target topics of our experiments, for Cancer, four related categories, Disease, Medicine, Oncology and Health, are chosen from the category hierarchy. For Fiction, there are two related categories, Literature and Arts. For Olympics, one related category, Sports, is selected, for Cameras, three categories, Photography, Movies, and Arts, for Geology, two categories, Earth and Science, and for Poetry, two related categories, Literature and Arts. A state is a eight + $2 \cdot \alpha$ dimensional vector where α is the number of categories selected from a category hierarchy. Like a state, an action is represented as a six + $2 \cdot \alpha$ dimensional feature vector.

In this section, we compare our three proposed crawling algorithms based on reinforcement learning (synchronous method, asynchronous method, and asynchronous method with moderated update), and an algorithm without learning. The no-learning algorithm is served as a baseline of performance. It uses w2v based cosine similarity as a priority in the crawling frontier and does not use any features or learning update

formulas presented in Sec. 3. In no learning algorithm, when crawling a page, each outlinks is added to the frontiers with its w2v based cosine similarity based on the URL text, the anchor text and surrounding text of the hyperlink. Then, a link with the highest priority is selected for the next crawling.



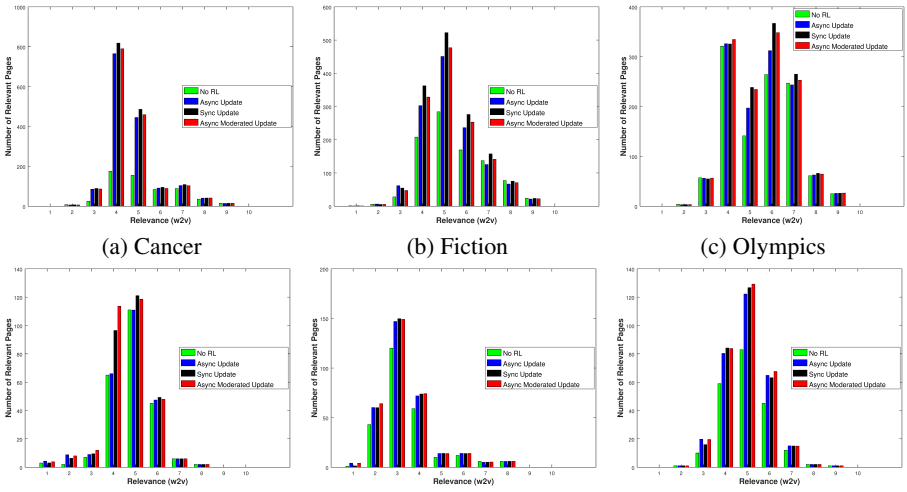
(a) Cancer (b) Fiction (c) Olympics

(d) Cameras (e) Geology (f) Poetry

Fig. 1. Accumulated Number of Relevant Pages per Time Step

Given a crawling task that visit 10,000 pages, Fig. 1 shows the accumulated number of relevant pages per time step during the crawling task. The x axis represents time step of a crawling task and the y axis marks the accumulated number of relevant pages per time step. Each curve is the average of 100 tasks. For each task, all data obtained during a crawling task is initialized, for example, hyperlinks in the frontier and parent information of hyperlinks, etc., but the weight vector learned in the precedent task is maintained. For all target topics, the algorithm without learning finds relevant pages progressively as time steps increase. For some topics such as Olympics, Cameras, Geology, and Poetry, we can see a sharp increase in early time steps. This is because a given seed page is a main page corresponding to each target topic, thus, the agent has more chance to meet relevant pages in early time steps. Compared to no learning with monotonous increase, reinforcement learning algorithms speed up finding relevant pages. In particular, for topic Cancer, Fiction, Geology, and Poetry, the accumulated number of relevant pages is increased abruptly. It means that reinforcement learning effectively helps find relevant pages as time steps increase. For topic Olympics and Cameras, the agent based on reinforcement learning follows a similar curve as no learning but finds more relevant pages.

Fig. 2 displays the quality of the experimental results above. The x axis marks w2v-based relevance discretized into 10 intervals and the y axis represents the number of relevant pages per relevance level. Each bar is the average of 100 tasks. In lower or higher levels of relevance, there is no significant difference among all algorithms



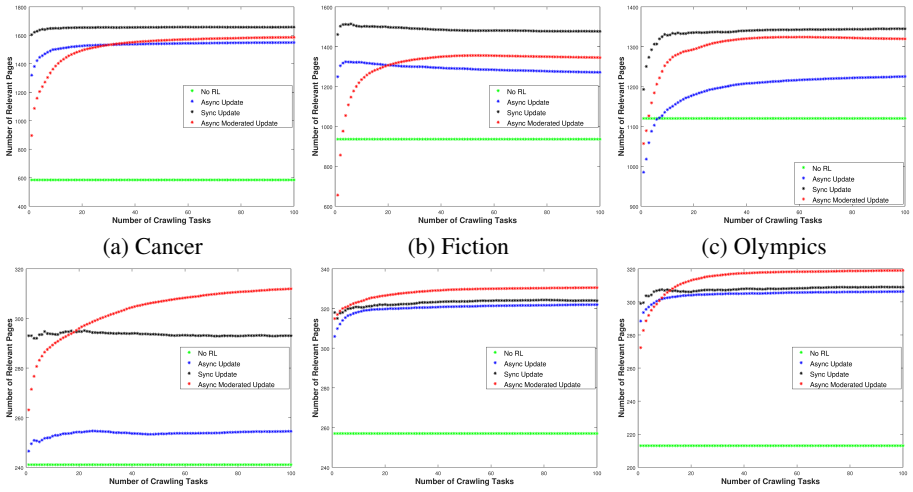
(a) Cancer (b) Fiction (c) Olympics

(d) Cameras (e) Geology (f) Poetry

Fig. 2. Number of Relevant Pages per Relevance Interval

because there are not many pages corresponding to those relevances on the environment. Meanwhile, it is apparent that learning and no learning algorithms have a big difference of performance for 3rd to 6th relevance levels depending on the distribution of relevant pages on the Web graph for each topic. Among learning algorithms, their performance results are similar or slightly different depending on topics. For topic Cancer, Geology, and Poetry, learning algorithms find similar number of relevant pages per relevant level. For topic Fiction, Olympics and Cameras, we can see a bit difference of performance between learning algorithms.

Fig. 3 shows learning curves of three algorithms on the different target topics. Each curve is the average of 10 independent trials. The x axis represents the number of crawling tasks and the y axis marks the number of relevant pages per task. A crawling task consists of visiting 10,000 pages. The learning curves show how the learning is improved as a task repeats. For each task, the weight vector learned in the precedent task is maintained and all other data obtained during a crawling task is initialized, for example, hyperlinks in the frontier and parent information of hyperlinks, etc. The same seed pages are given for each task. Thus, each crawling task starts in the same condition except the weight vector. By the learning curves, we can see how a crawling task is improved given the same condition. We compare reinforcement learning algorithms with no learning algorithm. Since each task is executed under the same condition, no learning algorithm's performance is the same regardless of the number of crawling tasks. For all target topics, reinforcement learning algorithms have better performance than the algorithm without learning. Those performances are generally 1.2 to 1.6 times, in particular, for topic Cancer, 2.5 times better than that of no learning algorithm. In Fig. 3(a)–(c), among all reinforcement learning algorithms, the synchronous method has the highest performance. In asynchronous methods, the moderated update outperforms the original update. In Fig. 3(d)–(f), the moderated update finds relevant pages more than



(a) Cancer (b) Fiction (c) Olympics

(d) Cameras (e) Geology (f) Poetry

Fig. 3. Number of Relevant Pages as Tasks Repeat

the other algorithms. Relevant pages of the three topics exist sparsely on the environment and thus those topics need more exploration. Since action values of unvisited links in the frontier are calculated at different time steps, those different values can hinder a good selection. By the moderated method, we can reduce the action value differences between time steps and effectively explore promising links while being less influenced by time steps.

From Fig. 3, we see that the synchronous method is better in general but the overhead of updating all action values cannot be ignored. For example, the computation time of synchronous method is 654 seconds while that of asynchronous and no learning are 55 and 28 seconds respectively for one crawling task of topic Olympics. Thus, if we consider the overhead of updates, the asynchronous method with moderated update can be a good alternative and may be even better in the environment in which the agent needs more exploration and in which action value differences are influenced by time steps.

5 Related Work

Chakrabarti et al. [4] first introduced focused crawling to selectively seek out pages that are relevant to a pre-defined set of topics, using both a classifier and a distiller, to guide the crawler. Diligenti et al. [10] introduced a context-focused crawler that improves on traditional focused crawling. It is trained by a context graph with multiple layers and used to estimate the link distance of a crawled page from a set of target pages.

The intelligent crawler [1] proposed by Aggarwal et al. statistically learns the characteristics of the link structure of the Web during the crawl. Using this learned statistical model, the crawler gives priorities to URLs in the frontier. Almpanidis et al. [2] propose a latent semantic indexing classifier that combines link analysis and content information. Chau et al. [5] focus on how to filter irrelevant documents from a set of documents

collected from the Web, through a classification that combines Web content analysis and Web structure analysis.

A few previous works have applied reinforcement learning to focused crawling. Rennie and McCallum [22] first use reinforcement learning in Web crawling, though their algorithm requires off-line training and action values are stored in a tabular form. Grigoriadis et al. [13] use a gradient descent function approximation method based on neural networks to estimate state values but, again, their algorithm also needs training phase before crawling task. The algorithm does not estimate link scores directly but links inherit their parent's score to use as priority values. InfoSpiders [16, 17] uses a neural network to estimate links. The neural net learns action values of links on-line, but does not use a function approximation method. In those methods, a state is represented with a vector which consists of the existence or frequency of specific keywords. Our method uses a generalized representation for states and actions by extracting features from pages and links, and learns action values in an online and incremental manner as traditional crawling agents do.

Meusel et al. [18] combine online classification and bandit-based selection strategy. To select a page to be crawled, they first use the bandit-based approach to choose a host with the highest score. Then, a page from the host is taken using the online classifier. Similarly, Gouriten et al. [12] use bandits to choose estimators for scoring the frontier.

Like reinforcement learning, crawling involves a trade-off between exploration and exploitation of information: greedily visiting URLs that have high estimate scores vs exploring URLs that seem less promising but might lead to more relevant pages and increase overall quality of crawling. Pant et al. [21] demonstrate that the best-N-first outperforms the naive best-first. The best-N-first algorithm picks and fetches top N links from the frontier for each iteration of crawling.

6 Conclusion

In this study, we applied reinforcement learning to focused crawling. We propose new representations for Web pages and next link selection using contextual information and the link structure. A number of pages and links are generalized with the proposed features. Based on this generalization, we used a linear function approximation with gradient descent to score links in the frontier. We investigated the trade-off between synchronous and asynchronous methods. As an improved asynchronous method, we propose moderated update to reach a balance between action-values updated at different time steps. Experimental results showed that reinforcement learning allows to estimate long-term link scores and to efficiently crawl relevant pages. In future work, we hope to evaluate our method in larger and various datasets, such as full English Wikipedia and dataset from the site http://commoncrawl.org/, etc. Another challenging possibility is to build up an efficient mechanism for categories selection to avoid a system designer pre-selecting proper categories for each target topic. Finally, We also want to investigate other ways to deal with exploration/exploitation.

References

1. C. C. Aggarwal, F. Al-Garawi, and P. S. Yu. Intelligent crawling on the World Wide Web with arbitrary predicates. In *WWW*, 2001.
2. G. Almpanidis, C. Kotropoulos, and I. Pitas. Combining text and link analysis for focused crawling. An application for vertical search engines. *Inf. Syst.*, 32(6), 2007.
3. D. Bergmark, C. Lagoze, and A. Sbityakov. Focused crawls, tunneling, and digital libraries. In *ECDL*, 2002.
4. S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: A new approach to topic-specific web resource discovery. In *WWW*, 1999.
5. M. Chau and H. Chen. A machine learning approach to web page filtering using content and structure analysis. *Decis. Support Syst.*, 44(2), 2008.
6. P. Dai and J. Goldsmith. Topological value iteration algorithm for Markov decision processes. In *IJCAI*, 2007.
7. P. Dai and E. A. Hansen. Prioritizing Bellman backups without a priority queue. In *ICAPS*, 2007.
8. B. D. Davison. Topical locality in the web. In *SIGIR*, 2000.
9. J. S. Dibangoye, B. Chaib-draa, and A.-i. Mouaddib. A novel prioritization technique for solving Markov decision processes. In *FLAIRS*, 2008.
10. M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, and M. Gori. Focused crawling using context graphs. In *VLDB*, 2000.
11. M. Ester, M. Groß, and H.-P. Kriegel. Focused web crawling: A generic framework for specifying the user interest and for adaptive crawling strategies. In *VLDB*, 2001.
12. G. Gouriten, S. Maniu, and P. Senellart. Scalable, generic, and adaptive systems for focused crawling. In *HyperText*, pages 35–45, 2014.
13. A. Grigoriadis and G. Paliouras. Focused crawling using temporal difference-learning. In G. A. Vouros and T. Panayiotopoulos, editors, *SETN*, 2004.
14. J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5), 1999.
15. F. Menczer. Lexical and semantic clustering by web links. *J. Am. Soc. Inf. Sci. Technol.*, 55(14), 2004.
16. F. Menczer and R. K. Belew. Adaptive retrieval agents: Internalizing local contextand scaling up to the web. *Mach. Learn.*, 39(2-3), 2000.
17. F. Menczer, G. Pant, and P. Srinivasan. Topical web crawlers: Evaluating adaptive algorithms. *ACM Trans. Internet Technol.*, 4(4), 2004.
18. R. Meusel, P. Mika, and R. Blanco. Focused crawling for structured data. In *CIKM*, 2014.
19. A. W. Moore and C. G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Mach. Learn.*, 13(1), 1993.
20. L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, 1999.
21. G. Pant, P. Srinivasan, and F. Menczer. Exploration versus exploitation in topic driven crawlers. In *WWW Workshop on Web Dynamics*, 2002.
22. J. Rennie and A. McCallum. Using reinforcement learning to spider the web efficiently. In *ICML*, 1999.
23. R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1998.
24. D. Wingate and K. D. Seppi. P3VI: A partitioned, prioritized, parallel value iterator. In *ICML*, 2004.
25. D. Wingate and K. D. Seppi. Prioritization methods for accelerating MDP solvers. *J. Mach. Learn. Res.*, 6, 2005.